Université Bordeaux 1
Licence, module Compilation, 2016/2017

TD2-Analyse lexicale

Chaque exercice consiste à réaliser un programme exécutable `exoi.o` en utilisant le compilateur d'analyseur lexical flex.

Exercice 2.1

Appliquer, sur tout texte écrit en Français, sans accents, les substitutions suivantes :
Alpes → Pyrenees, loup → ours, loups → ours, chamois → {izard, izards}, Pralognan → Saint − Girons
en essayant de respecter l'accord en nombre (singulier/pluriel).
Attention : "loupe", "loupiotte", "chevre chamoisee" restent invariants.
Par exemple :

```
Les loups et les brebis manifestent, chacun de leur cote, dans les rues de Pralognan.
Quelques chamois ainsi que des chevres chamoisees se sont joint aux brebis,
par solidarite sans doute. Un vieux chamois sage a accepte d'etre nomme mediateur.
```

sera transformé en

```
Les ours et les brebis manifestent, chacun de leur cote, dans les rues de Saint-Girons.
Quelques izards ainsi que des chevres chamoisees se sont joint aux brebis,
par solidarite sans doute. Un vieux izard sage a accepte d'etre nomme mediateur.
```

Exercice 2.2

Conjuguer un verbe du premier groupe au présent de l'indicatif. La donnée sera de la forme $< pronom\ personnel > [verbe] < complements\ eventuels >$.
Par exemple : sur `ils [trouver] des pommes`, le programme devra retourner

```
ils trouvent des pommes
```

et sur `je [marcher] sur la plage`, le programme devra retourner

```
je marche sur la plage
```

Exercice 2.3

Reconnaître les identificateurs de variables dans un programme IMP. En fin d'analyse : écrire les couples *idvar* : *nocc* où *nocc* est le nombre d'occurrences de l'identificateur *idvar* dans le programme.
Attention : `while, do, if, then, else`, etc ... ne sont pas des identificateurs. Exemple : sur l'entrée `(((X1 := 2+LONG ; X1 := X1 + 3);X2 := 10) ; while X2 do (X1 := X1 *2))` la sortie sera `(X2 : 2) (LONG : 1) (X1 : 5)`

Exercice 2.4

On suppose qu'une définition de fonction est de la forme :

```
defun idfon(idarg1:type1, ...,idargn:typen)
```

où les types sont de la forme :

```
type ::= int |  bool | array of type
```

Tester que le nombre d'arguments de l'appel d'une fonction, sur des arguments atomiques (identificateur de variable ou de constante ou numeral) est conforme à sa déclaration.
Quelle est la difficulté rencontrée lorsqu'on essaye d'étendre ce test à des arguments qui sont des expressions quelconques ?

Exercice 2.5

Le professeur Cosinus [1] a l'habitude de truffer ses programmes C de suites d'instructions "de mise au point" de la forme :

```
/* begindebug
inst1
inst2
...
enddebug */
```

Ecrire une commande `nettoyer.o` qui prend en entrée un programme C de Cosinus et retourne une version de ce programme débarassée des suites d'instructions de mise au point.

1. qui maîtrise mal la commande `gdb`

Exercice 2.6
Un programme écrit dans le langage C3A (Code à 3 Adresses) consiste en une suite de lignes, chacune de la forme :

$$Etiquette : Operateur : Argument : Argument : Destination$$

Les champs Etiquette, Destination sont des identificateurs i.e. des mots qui consistent en une lettre suivie d'un nombre quelconque de lettres ou chiffres. Operateur est l'une des chaînes de caractères :

$$\mathrm{Pl}, \mathrm{Mo}, \mathrm{Mu}, \mathrm{Af}, \mathrm{Afc}, \mathrm{Sk}, \mathrm{Jp}, \mathrm{Jz}, \mathrm{St}$$

et Argument peut être soit un identificateur, soit un numéral i.e. une suite de chiffres, précédée éventuellement d'un signe ("+" ou "-").
La sémantique de ce langage est définie (informellement) comme suit :
- Pl affecte à la variable Destination la somme du premier et du second argument
- Mo affecte à la variable Destination la différence du premier et du second argument
- Mu affecte à la variable Destination le produit du premier et du second argument
- Af provoque une affectation de la valeur du second argument au premier argument
- Afc provoque une affectation de la valeur du premier argument, qui est un numeral, à la variable Destination
- Sk ne provoque aucune modification de l'environnement
- Jp provoque un saut à l'instruction étiquetée par Destination
- Jz provoque un saut à l'instruction étiquetée par Destination, dans le cas où le premier argument est nul
- St arrête l'exécution.

En fait, vu cette sémantique, seul le champ Operateur doit impérativement être correctement rempli ; les autres champs peuvent être "vides" c'est à dire consister en une suite de caractères " blanc" ou "tabulation", lorsque l'opération (de la même ligne) ne les utilise pas et ils ne sont pas la destination d'une instruction de saut (d'une autre ligne).
1- Ecrire une commande qui, prenant un entrée un programme C3A, construit un objet C qui est une liste chaînée de quadruplets au sens du type `BILQUAD` :

```
char *ETIQ;
int  OP;
char *ARG1, *ARG2, *RES;
struct cellquad *SUIV;} *QUAD;

typedef struct{
QUAD debut;
QUAD fin;}BILQUAD;
```

2- Écrire une commande qui interprète le programme C3A sur l'environnement initial nul (chaque variable a une valeur initiale nulle).

Des fonctions auxiliaires maniant les environnements et les quadruplets sont disponibles dans les fichiers `environ.c` et `bilquad.c`.
Exemple : l'interprétation du programme

```
ET2       :Sk     :          :          : X2
ET3       :Afc    :10        :          : CT4
ET1       :Pl     :X2        :CT4       : VA5
ET0       :Af     :X1        :VA5       :
ET6       :Sk     :          :          :
ET9       :Sk     :          :          : X1
ET10      :Afc    :3         :          : CT11
ET8       :Mu     :X1        :CT11      : VA12
ET7       :Af     :X1        :VA12      :
ET13      :St     :          :          :
```

produira l'environnement :
`(VA12:30) (CT11:3) (VA5:10) (X1:30) (CT4:10) (X2:0)`